1.Plugin development

✅ Step-by-Step Guide to Create a Basic Plugin

1. Create Plugin Folder and File

Go to /wp-content/plugins/ and create a folder, e.g., my-custom-plugin.

Inside it, create a PHP file, e.g., my-custom-plugin.php.

2. Add Plugin Header

<?php

/\*

Plugin Name: My Custom Plugin

Description: A simple custom plugin example.

Version: 1.0

Author: Your Name

\*/

#### 3. **Write Your Plugin Logic**

Example: Add a message to the footer of every page.

add\_action('wp\_footer', 'custom\_footer\_message');

function custom\_footer\_message() {

echo "<p style='text-align:center;'>Thank you for visiting!</p>";

}

2. Custom Gutenberg blocks?

## 🔧 What Is a Gutenberg Block?

A block is a content element in the WordPress block editor. You can create:

* Text blocks
* Image/text layout blocks
* Sliders
* Testimonials
* Product features
* …anything!

🧱 How to Create a Custom Gutenberg Block

### ✅ Step 1: **Setup Plugin Folder**

Create a folder:

/wp-content/plugins/my-custom-block/

Add these files:

my-custom-block.php

block/

├── index.js

├── editor.css

└── style.css

✅ Step 2: my-custom-block.php

<?php

/\*\*

\* Plugin Name: My Custom Block

\*/

function my\_custom\_block\_register() {

wp\_register\_script(

'my-custom-block-editor',

plugins\_url('block/index.js', \_\_FILE\_\_),

['wp-blocks', 'wp-element', 'wp-editor', 'wp-components', 'wp-i18n'],

filemtime(plugin\_dir\_path(\_\_FILE\_\_) . 'block/index.js')

);

wp\_register\_style(

'my-custom-block-editor-style',

plugins\_url('block/editor.css', \_\_FILE\_\_),

[],

filemtime(plugin\_dir\_path(\_\_FILE\_\_) . 'block/editor.css')

);

wp\_register\_style(

'my-custom-block-style',

plugins\_url('block/style.css', \_\_FILE\_\_),

[],

filemtime(plugin\_dir\_path(\_\_FILE\_\_) . 'block/style.css')

);

register\_block\_type('myplugin/custom-block', [

'editor\_script' => 'my-custom-block-editor',

'editor\_style' => 'my-custom-block-editor-style',

'style' => 'my-custom-block-style',

]);

}

add\_action('init', 'my\_custom\_block\_register');

✅ Step 3: block/index.js (React-style JSX)

const { registerBlockType } = wp.blocks;

const { RichText } = wp.blockEditor;

registerBlockType('myplugin/custom-block', {

title: 'My Custom Block',

icon: 'smiley',

category: 'widgets',

attributes: {

content: { type: 'string' }

},

edit: (props) => {

const { attributes: { content }, setAttributes } = props;

return (

<div className="my-custom-block">

<RichText

tagName="p"

value={content}

onChange={(value) => setAttributes({ content: value })}

placeholder="Enter your text..."

/>

</div>

);

},

save: (props) => {

return (

<div className="my-custom-block">

<RichText.Content tagName="p" value={props.attributes.content} />

</div>

);

}

});

✅ Step 4: CSS Files

block/editor.css

.my-custom-block {

border: 2px dashed #999;

padding: 10px;

background: #fffefc;

}

block/style.css

.my-custom-block {

background: #f9f9f9;

padding: 15px;

font-size: 18px;

}

3.Theme customization

## 🎯 What do you want to customize?

Here are common theme customizations:

| Task | Tools/Method |
| --- | --- |
| Change colors, fonts, spacing | CSS / Customizer |
| Add sections (e.g., testimonials, banner, etc.) | Modify theme files / use page builders |
| Add custom templates (e.g., for blog, category, product) | template-{type}.php |
| Add dynamic fields (ACF or custom PHP) | Use Advanced Custom Fields or get\_post\_meta() |
| Customize header/footer | header.php, footer.php |
| Override parent theme safely | Use a child theme |
| Add Gutenberg block patterns | Register in functions.php |

## 🛠️ Do You Have a Specific Use Case?

Please pick one:

* ✅ I want to create a custom homepage layout.
* ✅ I want to edit header or footer.
* ✅ I want to add dynamic fields (like ACF or meta boxes).
* ✅ I want to create a child theme to avoid losing changes.
* ✅ I want to convert an HTML design into a theme.
* ✅ Something else (please describe).

## 🧪 Example: Customize Homepage

If your theme’s homepage is front-page.php or home.php, you can customize it like this:

<?php get\_header(); ?>

<section class="hero">

<h1>Welcome to My Site</h1>

<p><?php bloginfo('description'); ?></p>

</section>

<section class="latest-posts">

<h2>Latest Posts</h2>

<?php

$posts = new WP\_Query(['posts\_per\_page' => 3]);

while ($posts->have\_posts()): $posts->the\_post(); ?>

<article>

<h3><?php the\_title(); ?></h3>

<p><?php the\_excerpt(); ?></p>

<a href="<?php the\_permalink(); ?>">Read More</a>

</article>

<?php endwhile; wp\_reset\_postdata(); ?>

</section>

<?php get\_footer(); ?>

## 🎨 Tip: Add Custom CSS

Go to:  
Appearance → Customize → Additional CSS  
Or create a style.css file in your child theme.

4.create custom theme

## 🎯 What You’ll Get:

* Theme folder structure
* style.css with theme info
* index.php, header.php, footer.php
* Navigation, blog loop, sidebar support
* Enqueued CSS & JS
* Ready to customize layout

📁 Theme Folder Structure

/wp-content/themes/my-custom-theme/

my-custom-theme/

├── style.css

├── index.php

├── header.php

├── footer.php

├── functions.php

├── sidebar.php

└── assets/

├── css/

│ └── main.css

└── js/

└── main.js

✅ 1. style.css

/\*

Theme Name: My Custom Theme

Theme URI: https://example.com

Author: Your Name

Author URI: https://yourwebsite.com

Description: A simple custom theme built from scratch.

Version: 1.0

License: GPL-2.0+

Text Domain: my-custom-theme

\*/

✅ 2. functions.php

<?php

function mytheme\_setup() {

add\_theme\_support('title-tag');

add\_theme\_support('post-thumbnails');

register\_nav\_menus([

'main\_menu' => 'Main Menu'

]);

}

add\_action('after\_setup\_theme', 'mytheme\_setup');

function mytheme\_enqueue\_assets() {

wp\_enqueue\_style('mytheme-style', get\_stylesheet\_directory\_uri() . '/assets/css/main.css');

wp\_enqueue\_script('mytheme-script', get\_stylesheet\_directory\_uri() . '/assets/js/main.js', [], null, true);

}

add\_action('wp\_enqueue\_scripts', 'mytheme\_enqueue\_assets');

✅ 3. header.php

<!DOCTYPE html>

<html <?php language\_attributes(); ?>>

<head>

<meta charset="<?php bloginfo('charset'); ?>">

<?php wp\_head(); ?>

</head>

<body <?php body\_class(); ?>>

<header>

<h1><a href="<?php echo home\_url(); ?>"><?php bloginfo('name'); ?></a></h1>

<nav>

<?php wp\_nav\_menu(['theme\_location' => 'main\_menu']); ?>

</nav>

</header>

✅ 4. footer.php

<footer>

<p>&copy; <?php echo date('Y'); ?> <?php bloginfo('name'); ?></p>

</footer>

<?php wp\_footer(); ?>

</body>

</html>

✅ 5. index.php

<?php get\_header(); ?>

<main>

<?php if (have\_posts()): while (have\_posts()): the\_post(); ?>

<article>

<h2><?php the\_title(); ?></h2>

<div><?php the\_excerpt(); ?></div>

</article>

<?php endwhile; else: ?>

<p>No posts found.</p>

<?php endif; ?>

</main>

<?php get\_sidebar(); ?>

<?php get\_footer(); ?>

✅ 6. sidebar.php

<aside>

<?php if (is\_active\_sidebar('main\_sidebar')): ?>

<?php dynamic\_sidebar('main\_sidebar'); ?>

<?php else: ?>

<p>Add widgets to Sidebar</p>

<?php endif; ?>

</aside>

Add this to functions.php to register the sidebar:

function mytheme\_widgets\_init() {

register\_sidebar([

'name' => 'Main Sidebar',

'id' => 'main\_sidebar',

'before\_widget' => '<div class="widget">',

'after\_widget' => '</div>',

'before\_title' => '<h3>',

'after\_title' => '</h3>',

]);

}

add\_action('widgets\_init', 'mytheme\_widgets\_init');

✅ 7. main.css (Basic)

body {

font-family: sans-serif;

margin: 0;

padding: 0;

}

header, footer {

background: #333;

color: #fff;

padding: 1em;

}

main {

padding: 2em;

}

✅ 8. main.js (Placeholder)

console.log("Theme JS loaded");

✅ Final Step: Activate the Theme

Go to Appearance → Themes and activate My Custom Theme.

5.hooks in wordpress

## 🔄 Two Types of Hooks

| Hook Type | Purpose | Example |
| --- | --- | --- |
| Actions | Perform operations (e.g., send email, enqueue scripts) | add\_action('wp\_head', 'my\_function') |
| Filters | Modify data (e.g., change post title, modify content) | add\_filter('the\_content', 'my\_filter\_function') |

🔧 1. Actions

👉 Example: Add custom code to <head>

add\_action('wp\_head', 'my\_custom\_head\_code');

function my\_custom\_head\_code() {

echo '<meta name="author" content="Your Name">';

}

👉 Example: Run code after post is saved

add\_action('save\_post', 'log\_post\_save');

function log\_post\_save($post\_id) {

error\_log("Post $post\_id was saved!");

}

🧪 2. Filters

add\_filter('the\_title', 'custom\_post\_title');

function custom\_post\_title($title) {

return '🔥 ' . $title;

}

👉 Example: Modify content before display

add\_filter('the\_content', 'append\_custom\_content');

function append\_custom\_content($content) {

if (is\_single()) {

$content .= '<p><em>Thanks for reading!</em></p>';

}

return $content;

}

✅ Use do\_action() and apply\_filters() in themes/plugins:

do\_action('my\_custom\_action');

apply\_filters('my\_custom\_filter', $value);

🧰 Create Your Own Hook in a Plugin/Theme

Action Hook

do\_action('my\_custom\_hook');

add\_action('my\_custom\_hook', function() {

echo 'Hello from custom action!';

});

Filter Hook

$value = apply\_filters('modify\_price', 100);

add\_filter('modify\_price', function($price) {

return $price \* 0.9; // Apply 10% discount

});

## 🚀 Most Common Hooks You Should Know

| Hook | Type | Purpose |
| --- | --- | --- |
| init | Action | Bootstrap code |
| wp\_enqueue\_scripts | Action | Enqueue styles/scripts |
| the\_content | Filter | Modify post content |
| wp\_footer / wp\_head | Action | Inject code into theme |
| save\_post | Action | Do something after saving a post |
| excerpt\_more | Filter | Change excerpt “read more…” |
| login\_redirect | Filter | Change login redirect URL |

6.taxonomy

In WordPress, a taxonomy is a way to group and classify content — just like categories and tags group blog posts.

You can also create your own custom taxonomies to organize custom post types like products, events, books, etc.

## 🧠 Built-in Taxonomies

| Taxonomy | Applies To | Description |
| --- | --- | --- |
| category | Posts | Hierarchical (like folders) |
| post\_tag | Posts | Non-hierarchical (like labels) |

## ✅ Register a Custom Taxonomy

Example: Create a taxonomy called "Genres" for a book post type.

function create\_genre\_taxonomy() {

register\_taxonomy('genre', 'book', [

'labels' => [

'name' => 'Genres',

'singular\_name' => 'Genre',

],

'public' => true,

'hierarchical' => true, // like categories (true), or tags (false)

'show\_in\_rest' => true, // show in Gutenberg

'rewrite' => ['slug' => 'genre'],

]);

}

add\_action('init', 'create\_genre\_taxonomy');

📦 Attach Taxonomy to a Post Type

register\_taxonomy('brand', ['product'], [

'label' => 'Brand',

'hierarchical' => false,

'show\_in\_rest' => true,

]);

Now you can:

* Assign "brands" to products
* Filter products by brand
* Use get\_the\_terms() or wp\_get\_post\_terms() to fetch them

🧪 Display Taxonomy Terms in a Template

Show all terms of a taxonomy:

$terms = get\_terms('genre');

foreach ($terms as $term) {

echo '<a href="' . get\_term\_link($term) . '">' . $term->name . '</a><br>';

}

Show terms for a specific post:

$terms = get\_the\_terms(get\_the\_ID(), 'genre');

if ($terms && !is\_wp\_error($terms)) {

foreach ($terms as $term) {

echo '<span>' . $term->name . '</span>';

}

}

## 🧱 Add Taxonomy Field to Gutenberg or Classic Editor

It will appear automatically in the sidebar for your post type if:

* 'show\_ui' => true
* 'show\_in\_rest' => true (for Gutenberg)
* 'meta\_box\_cb' => true (for classic UI)

🔁 Use Taxonomies in Queries

$query = new WP\_Query([

'post\_type' => 'book',

'tax\_query' => [[

'taxonomy' => 'genre',

'field' => 'slug',

'terms' => 'science-fiction',

]]

]);

7.wordpress cli

The WordPress CLI (WP-CLI) is a powerful command-line tool to manage WordPress websites efficiently — no dashboard needed.

## ⚙️ What Can WP-CLI Do?

You can:

| Task | Command Example |
| --- | --- |
| Install WordPress | wp core install |
| Update core/plugins/themes | wp core update, wp plugin update --all |
| Manage users | wp user create, wp user delete |
| Manage posts | wp post create, wp post list |
| Manage DB | wp db export, wp db import |
| Manage themes/plugins | wp plugin install, wp theme activate |

🚀 Install WP-CLI (Ubuntu/Linux)

curl -O https://raw.githubusercontent.com/wp-cli/builds/gh-pages/phar/wp-cli.phar

php wp-cli.phar --info # Check if it works

chmod +x wp-cli.phar

sudo mv wp-cli.phar /usr/local/bin/wp

Now you can run:

wp –info

📦 Common WP-CLI Commands

✅ Install WordPress

wp core download

wp config create --dbname=wp\_db --dbuser=root --dbpass=root

wp db create

wp core install --url="http://localhost" --title="Site" --admin\_user=admin --admin\_password=admin --admin\_email=admin@example.com

✅ Plugin Management

wp plugin install contact-form-7 --activate

wp plugin deactivate akismet

wp plugin delete hello

✅ Theme Management

wp theme install twentytwentyfour --activate

wp theme list

✅ Create a New Post

wp post create --post\_title="Hello CLI" –post\_status=publish

✅ Create a User

wp user create editor user@example.com --role=editor –user\_pass=strongpass

✅ Export & Import DB

wp db export backup.sql

wp db import backup.sql

🧪 Advanced

Search & Replace in DB

wp search-replace 'oldurl.com' 'newurl.com'

Generate Dummy Content

wp post generate --count=50

wp user generate --count=10 –role=author

8.Create custom WP-CLI commands in plugins

Creating custom WP-CLI commands in a plugin lets you extend WordPress functionality directly from the terminal — great for automation, debugging, data import/export, etc.

✅ 1. Plugin Boilerplate

Create a new plugin file like wp-cli-mytools.php in wp-content/plugins/wp-cli-mytools/.

<?php

/\*\*

\* Plugin Name: WP CLI My Tools

\* Description: Custom WP-CLI commands.

\*/

// Only load in WP-CLI context

if (defined('WP\_CLI') && WP\_CLI) {

class My\_Custom\_CLI\_Commands {

/\*\*

\* Say hello to a user.

\*

\* ## OPTIONS

\* <name>

\* : The name to greet.

\*

\* ## EXAMPLES

\* wp mytools hello John

\*/

public function hello($args, $assoc\_args) {

list($name) = $args;

WP\_CLI::success("Hello, $name!");

}

/\*\*

\* Create 5 test posts.

\*

\* ## EXAMPLES

\* wp mytools create\_posts

\*/

public function create\_posts() {

for ($i = 1; $i <= 5; $i++) {

wp\_insert\_post([

'post\_title' => "CLI Post $i",

'post\_content' => "Content for CLI Post $i",

'post\_status' => 'publish',

'post\_type' => 'post',

]);

}

WP\_CLI::success("5 posts created.");

}

}

WP\_CLI::add\_command('mytools', 'My\_Custom\_CLI\_Commands');

}

✅ 2. Activate the Plugin

In WordPress admin or via WP-CLI:

wp plugin activate wp-cli-mytools

✅ 3. Use the Commands

wp mytools hello John

# Output: Success: Hello, John!

wp mytools create\_posts

# Output: Success: 5 posts created.

Accept optional arguments

public function greet($args, $assoc\_args) {

$name = $assoc\_args['name'] ?? 'Guest';

WP\_CLI::line("Welcome, $name");

}

Run with:

wp mytools greet –name=David

9. customize woo commerce

Customizing WooCommerce lets you tailor the store to fit your exact needs, from layout tweaks to new features.

Here’s a quick overview of common WooCommerce customizations and how you can achieve them:

## 1. **Override WooCommerce Templates in Your Theme**

WooCommerce templates control product pages, checkout, cart, etc.

* Copy the template file from  
  wp-content/plugins/woocommerce/templates/  
  to  
  wp-content/themes/your-theme/woocommerce/  
  keeping the folder structure.
* Example: To customize single product page:  
  Copy single-product.php or parts like content-single-product.php.
* Edit copied files safely without losing changes on plugin update.

## 2. **Use WooCommerce Hooks (Actions & Filters)**

WooCommerce has many hooks for injecting or modifying content.

### Example: Add text after product title

add\_action('woocommerce\_single\_product\_summary', 'add\_custom\_text\_after\_title', 6);

function add\_custom\_text\_after\_title() {

echo '<p>Custom text after the product title.</p>';

}

Example: Change “Add to Cart” button text

add\_filter('woocommerce\_product\_single\_add\_to\_cart\_text', function() {

return 'Buy Now';

});

## 3. **Add Custom Fields / Product Meta**

Use [Advanced Custom Fields (ACF)](https://www.advancedcustomfields.com/) or manually add custom product meta.

Example: Add a custom product field in admin and display it on the product page.

// Display custom field on product page

add\_action('woocommerce\_single\_product\_summary', 'show\_custom\_product\_field', 25);

function show\_custom\_product\_field() {

global $product;

$custom\_field = get\_post\_meta($product->get\_id(), '\_custom\_field\_key', true);

if ($custom\_field) {

echo '<p>Custom Field: ' . esc\_html($custom\_field) . '</p>';

}

}

## 4. **Customize Checkout Fields**

Modify checkout form fields to add, remove, or change.

Example: Remove company name field

add\_filter('woocommerce\_checkout\_fields', 'custom\_override\_checkout\_fields');

function custom\_override\_checkout\_fields($fields) {

unset($fields['billing']['billing\_company']);

return $fields;

}

Add a new field:

add\_filter('woocommerce\_checkout\_fields', 'add\_custom\_checkout\_field');

function add\_custom\_checkout\_field($fields) {

$fields['billing']['billing\_custom\_note'] = [

'type' => 'text',

'label' => 'Custom Note',

'required' => false,

'class' => ['form-row-wide'],

'priority' => 120,

];

return $fields;

}

// Save the custom field

add\_action('woocommerce\_checkout\_update\_order\_meta', 'save\_custom\_checkout\_field');

function save\_custom\_checkout\_field($order\_id) {

if (!empty($\_POST['billing\_custom\_note'])) {

update\_post\_meta($order\_id, '\_billing\_custom\_note', sanitize\_text\_field($\_POST['billing\_custom\_note']));

}

}

## 5. **Custom Product Query / Display**

Example: Show only products from a specific category on a page:

$args = [

'post\_type' => 'product',

'tax\_query' => [[

'taxonomy' => 'product\_cat',

'field' => 'slug',

'terms' => 'your-category-slug',

]],

];

$loop = new WP\_Query($args);

if ($loop->have\_posts()) {

while ($loop->have\_posts()) {

$loop->the\_post();

wc\_get\_template\_part('content', 'product');

}

wp\_reset\_postdata();

}

## 6. **Add Custom WooCommerce Shortcodes**

Example: Shortcode to display 3 latest products:

add\_shortcode('latest\_products\_3', function() {

ob\_start();

echo do\_shortcode('[products limit="3" orderby="date" order="DESC"]');

return ob\_get\_clean();

});

## 7. **Style WooCommerce**

Use your theme’s style.css or enqueue a custom CSS file targeting WooCommerce classes, e.g.:

.woocommerce ul.products li.product {

border: 1px solid #ddd;

padding: 10px;

}

10.rest api in wordpress

The WordPress REST API allows you to interact with your WordPress site programmatically — creating, reading, updating, and deleting content via HTTP requests in JSON format. It’s the backbone for headless WordPress, mobile apps, integrations, and more.

## 🔥 Core Concepts

* Endpoints: URLs representing WordPress data (posts, pages, users, etc.).
* HTTP methods: GET (read), POST (create), PUT/PATCH (update), DELETE (delete).
* Authentication: Needed for protected operations (POST, PUT, DELETE).

## 🚀 Common Endpoints

| Endpoint | Description | Example URL |
| --- | --- | --- |
| /wp-json/wp/v2/posts | Get all posts or create a new post | https://example.com/wp-json/wp/v2/posts |
| /wp-json/wp/v2/pages | Get pages |  |
| /wp-json/wp/v2/categories | Get categories |  |
| /wp-json/wp/v2/users | Get users (authenticated) |  |
| /wp-json/wp/v2/comments | Get comments |  |

📖 Example: Fetch Posts with cURL

curl https://example.com/wp-json/wp/v2/posts

## 📌 Register Custom REST API Endpoint

Add to your plugin or theme’s functions.php:

add\_action('rest\_api\_init', function () {

register\_rest\_route('myplugin/v1', '/hello/', [

'methods' => 'GET',

'callback' => function () {

return ['message' => 'Hello from custom endpoint!'];

},

]);

});

Now, GET https://example.com/wp-json/myplugin/v1/hello/ returns:

{"message":"Hello from custom endpoint!"}

## 🔐 Authentication Methods

* Cookie Authentication (default for logged-in users)
* Basic Authentication (for development; insecure over HTTP)
* OAuth or JWT Authentication (recommended for production and external apps)

Plugins like JWT Authentication help set this up.

## ⚙️ Example: Create Post via REST API (Authenticated)

Using Basic Auth with cURL:

curl -X POST https://example.com/wp-json/wp/v2/posts \

-u username:password \

-H "Content-Type: application/json" \

-d '{"title":"My New Post","content":"Post content","status":"publish"}'

## 🧩 Extend REST API for Custom Post Types or Meta

When registering CPT or taxonomies, add 'show\_in\_rest' => true to make them available in REST API.

Example:

register\_post\_type('book', [

'label' => 'Books',

'public' => true,

'show\_in\_rest' => true,

]);

To add custom fields (meta) to REST API, use:

register\_rest\_field('post', 'my\_custom\_field', [

'get\_callback' => function ($post) {

return get\_post\_meta($post['id'], '\_my\_custom\_field', true);

},

'update\_callback' => function ($value, $post) {

update\_post\_meta($post->ID, '\_my\_custom\_field', sanitize\_text\_field($value));

},

'schema' => [

'description' => 'My custom field',

'type' => 'string',

],

]);

11.xml parsing

Parsing XML means reading XML data and extracting information from it. In WordPress/PHP, you can parse XML easily using built-in PHP tools.

## 🛠 Common PHP Methods for XML Parsing

### 1. **SimpleXML** (Easy to use, good for small/medium XML)

$xmlString = '<books><book><title>Book One</title></book><book><title>Book Two</title></book></books>';

$xml = simplexml\_load\_string($xmlString);

foreach ($xml->book as $book) {

echo $book->title . "<br>";

}

You can also load from a file:

$xml = simplexml\_load\_file('path/to/file.xml');

2. DOMDocument (More powerful & flexible, complex XML)

$dom = new DOMDocument;

$dom->load('path/to/file.xml');

$books = $dom->getElementsByTagName('book');

foreach ($books as $book) {

$title = $book->getElementsByTagName('title')->item(0)->nodeValue;

echo $title . "<br>";

}

3. XMLReader (Efficient for very large XML files, reads XML node-by-node)

$reader = new XMLReader();

$reader->open('path/to/largefile.xml');

while ($reader->read()) {

if ($reader->nodeType == XMLReader::ELEMENT && $reader->name == 'book') {

$bookNode = $reader->expand();

$dom = new DOMDocument();

$domNode = $dom->importNode($bookNode, true);

$dom->appendChild($domNode);

$title = $dom->getElementsByTagName('title')->item(0)->nodeValue;

echo $title . "<br>";

}

}

$reader->close();

🔧 Parsing XML in WordPress

If you’re building a plugin or theme, you can use these PHP methods directly.

Example: Fetch XML from remote URL and parse with SimpleXML

$response = wp\_remote\_get('https://example.com/data.xml');

if (is\_wp\_error($response)) {

return; // handle error

}

$body = wp\_remote\_retrieve\_body($response);

$xml = simplexml\_load\_string($body);

foreach ($xml->item as $item) {

echo $item->title . '<br>';

}

### 1. **Explain the WordPress hook system: actions vs filters. How do you create your own custom hooks?**

WordPress Hooks are a system to allow developers to modify or add functionality without editing core files.

* Actions  
  Actions are hooks that allow you to execute custom functions at specific points during WordPress execution. They don’t modify data, they just perform tasks.

Example:

add\_action('wp\_footer', function() {

echo '<p>Custom footer text</p>';

});

Filters  
Filters allow you to intercept and modify data before it is used or output. You receive data, modify it, and return it.

Example:

add\_filter('the\_content', function($content) {

return $content . '<p>Appended text</p>';

});

Creating Custom Hooks:

* Custom Action Hook  
  Inside your code, you define a place where other functions can hook in:

do\_action('my\_custom\_action', $arg1, $arg2);

Other plugins/themes can add callbacks to this:

add\_action('my\_custom\_action', 'my\_callback', 10, 2);

function my\_callback($arg1, $arg2) {

// Do something

}

Custom Filter Hook  
Similarly, for filters:

$value = apply\_filters('my\_custom\_filter', $value, $arg);

Others can hook with:

add\_filter('my\_custom\_filter', 'my\_filter\_callback', 10, 2);

function my\_filter\_callback($value, $arg) {

// Modify and return $value

return $value . ' modified';

}

### 2. **What is the WordPress load order? How does the initialization process work?**

WordPress Load Order Overview:

* wp-config.php — Loads the configuration, database credentials, debug settings.
* wp-settings.php — Loads core WordPress files and sets up the environment.
* Plugin loading
  + Plugins are loaded after core files in the order they appear in the plugins folder.
* Themes loaded
  + The active theme is loaded after plugins.
* Hooks triggered during load  
  The initialization happens in phases with key actions fired, such as:
  + muplugins\_loaded — Must-use plugins loaded.
  + plugins\_loaded — All plugins loaded.
  + after\_setup\_theme — Theme setup (register menus, image sizes).
  + init — WordPress initialization, suitable for registering CPTs, taxonomies.
  + wp\_loaded — WordPress is fully loaded, but before headers are sent.

Common flow example:

1. wp-config.php defines constants.
2. wp-settings.php loads core files.
3. Must-use plugins loaded.
4. Regular plugins loaded, plugins\_loaded action fires.
5. Theme’s functions.php loads, after\_setup\_theme action fires.
6. init action fires — you hook here to add CPTs, enqueue scripts.
7. Query vars parsed and query run.
8. Template loaded, page generated.

### 3. **How does the WordPress database schema work internally? What tables are crucial, and how are relationships handled?**

WordPress uses a MySQL database with a standard set of tables:

| Table Name | Purpose |
| --- | --- |
| wp\_posts | Stores all content types (posts, pages, CPTs) |
| wp\_postmeta | Metadata for posts (custom fields) |
| wp\_terms | Stores taxonomy terms (categories, tags) |
| wp\_term\_taxonomy | Defines taxonomy type for terms |
| wp\_term\_relationships | Links posts to taxonomy terms |
| wp\_users | User accounts |
| wp\_usermeta | Metadata for users |
| wp\_options | Site-wide options and settings |
| wp\_comments | Comments on posts |
| wp\_commentmeta | Metadata for comments |

Relationships:

* Posts and postmeta have a one-to-many relationship — one post, many meta key-value pairs.
* Terms are linked to posts via the term\_relationships table (many-to-many).
* Taxonomies (category, tag, custom taxonomies) define the relationship type stored in term\_taxonomy.
* Users and usermeta have a one-to-many relationship.

### 4. **How do you debug WordPress performance issues and memory leaks?**

Performance Debugging:

* Use Query Monitor plugin — shows slow queries, hooks, HTTP API calls.
* Enable WP\_DEBUG and SAVEQUERIES in wp-config.php:

define('WP\_DEBUG', true);

define('SAVEQUERIES', true);

Check slow database queries and optimize with indexes or caching.

* Profile PHP performance with tools like Xdebug, Blackfire, or New Relic.
* Use caching layers (object cache, page cache, CDN).
* Analyze front-end performance (Google Lighthouse, GTmetrix).

Memory Leak Debugging:

* Monitor PHP memory usage with memory\_get\_usage() or tools like New Relic.
* Increase memory\_limit in php.ini or wp-config.php if necessary.
* Profile and inspect custom plugins or themes that may hold references preventing garbage collection.
* Check for infinite loops or recursive hooks.
* Use WP-CLI commands to monitor memory during operations.
* Disable plugins/themes selectively to isolate problematic code.

### 1. **Walk me through creating a complex WordPress plugin from scratch with multiple features**

Step-by-step:

* Planning:  
  Define plugin goals, features, data models, and user roles.
* Setup Plugin Boilerplate:
  + Create plugin folder (e.g., my-plugin/)
  + Create main PHP file with plugin header (my-plugin.php)
  + Organize folders: /includes, /admin, /public, /assets
* Define Core Plugin Class:  
  Encapsulate all plugin logic inside a class to avoid global namespace pollution.
* Register Activation/Deactivation Hooks:  
  Setup database tables, default options on activation; cleanup on deactivation/uninstall.
* Create Admin Pages:  
  Use WordPress Settings API or custom UI with React (for complex cases).
* Custom Post Types/Taxonomies:  
  Register if needed with 'show\_in\_rest' => true for REST API.
* Add Shortcodes and Widgets:  
  For front-end display and flexibility.
* Implement REST API Endpoints:  
  For external integration or AJAX-based features.
* Use AJAX for Dynamic Features:  
  Hook AJAX actions properly (wp\_ajax\_ and wp\_ajax\_nopriv\_).
* Enqueue Scripts and Styles:  
  Properly load assets only on plugin pages or where needed.
* Security:  
  Use nonces, sanitize/validate inputs, check user permissions.
* Localization:  
  Make plugin translation-ready with load\_plugin\_textdomain().
* Documentation and Code Comments:  
  Write clear docs and comments for maintainability.

### 2. **How do you structure a large plugin for scalability and maintainability?**

Recommended structure:

my-plugin/

│

├── assets/

│ ├── css/

│ ├── js/

│ └── images/

│

├── includes/

│ ├── class-myplugin-activator.php

│ ├── class-myplugin-deactivator.php

│ ├── class-myplugin-admin.php

│ ├── class-myplugin-public.php

│ ├── class-myplugin-rest-api.php

│ └── functions.php

│

├── admin/

│ ├── views/

│ └── settings.php

│

├── public/

│ ├── views/

│ └── shortcode.php

│

├── languages/

│

└── my-plugin.php

Use OOP: Create classes for different responsibilities (Admin, Public, REST API).

* Autoloading: Use Composer’s PSR-4 autoload or a custom autoloader.
* Modular Code: Split features into separate classes/files.
* Hook Management: Register all hooks inside class methods.
* Configuration: Use constants or config files for paths and versioning.
* Error Handling: Use try/catch and proper logging.
* Version Control: Keep plugin code in Git for collaboration and rollback.

### 3. **How do you handle plugin activation, deactivation, and uninstall routines properly?**

* Activation:  
  Register activation hook and perform:
  + Create or update database tables.
  + Add default options/settings.
  + Schedule cron jobs if needed.

register\_activation\_hook(\_\_FILE\_\_, ['MyPlugin\_Activator', 'activate']);

Deactivation:  
Clean up temporary data, clear scheduled events, but don’t delete user data.

register\_deactivation\_hook(\_\_FILE\_\_, ['MyPlugin\_Deactivator', 'deactivate']);

Uninstall:  
Use uninstall.php or register uninstall hook to fully delete options, tables, and data.

register\_uninstall\_hook(\_\_FILE\_\_, ['MyPlugin\_Uninstaller', 'uninstall']);

Best practice: Never delete user data on deactivate; only on uninstall, and inform users.

### 4. **Explain security best practices for plugin development (nonce, sanitization, escaping, capabilities)**

* Nonces:  
  Use nonces to verify form submissions or AJAX requests to prevent CSRF.

wp\_nonce\_field('myplugin\_save\_action', 'myplugin\_nonce');

check\_admin\_referer('myplugin\_save\_action', 'myplugin\_nonce');

Sanitization:  
Clean all user inputs before saving to DB.

Examples:

$title = sanitize\_text\_field($\_POST['title']);

$email = sanitize\_email($\_POST['email']);

$url = esc\_url\_raw($\_POST['url']);

Escaping:  
Escape data when outputting to HTML to prevent XSS.

Examples:

echo esc\_html($title);

echo esc\_attr($attribute);

echo esc\_url($url);

Capabilities & Permissions:  
Check user capabilities before allowing any sensitive operation.

if (!current\_user\_can('manage\_options')) {

wp\_die(\_\_('You do not have permission'));

}

Use Prepared Statements  
Use $wpdb->prepare() for custom SQL queries to prevent SQL Injection.

* Avoid Direct Access:  
  Protect plugin files with:

defined('ABSPATH') || exit;

### 5. **How do you create custom WP-CLI commands for your plugin? Can you provide examples?**

* WP-CLI allows creating custom CLI commands to automate plugin tasks.

Example: Register a command inside your plugin main file or a CLI-specific class.

if (defined('WP\_CLI') && WP\_CLI) {

class MyPlugin\_CLI\_Command {

/\*\*

\* Says hello.

\*

\* ## EXAMPLES

\*

\* wp myplugin hello

\*

\* @when after\_wp\_load

\*/

public function hello($args, $assoc\_args) {

WP\_CLI::success("Hello from MyPlugin!");

}

/\*\*

\* Bulk update post meta.

\*

\* ## OPTIONS

\*

\* <post\_id>

\* : The ID of the post.

\*

\* <meta\_key>

\* : Meta key to update.

\*

\* <meta\_value>

\* : New value for the meta key.

\*

\* ## EXAMPLES

\*

\* wp myplugin update\_meta 123 \_my\_meta "New Value"

\*

\* @when after\_wp\_load

\*/

public function update\_meta($args) {

list($post\_id, $meta\_key, $meta\_value) = $args;

if (!update\_post\_meta($post\_id, $meta\_key, $meta\_value)) {

WP\_CLI::error("Failed to update meta.");

} else {

WP\_CLI::success("Meta updated for post $post\_id.");

}

}

}

WP\_CLI::add\_command('myplugin', 'MyPlugin\_CLI\_Command');

}

Usage:

* wp myplugin hello — prints hello message
* wp myplugin update\_meta 123 \_my\_meta "New Value" — updates post meta

### 1. **How do you create a fully custom theme from scratch? What files and hooks are essential?**

Steps to create a custom theme from scratch:

* Create Theme Folder:  
  Inside /wp-content/themes/, create a folder, e.g. my-custom-theme.
* Essential Files:
  + style.css  
    The main stylesheet with theme header comment:

/\*

Theme Name: My Custom Theme

Theme URI: http://example.com

Author: Your Name

Description: A custom WordPress theme

Version: 1.0

Text Domain: my-custom-theme

\*/

Recommended Additional Files:

* header.php — Site header, loaded via get\_header()
* footer.php — Site footer, loaded via get\_footer()
* sidebar.php — Sidebar widget area
* page.php — Template for pages
* single.php — Template for single posts
* archive.php — Template for archives (categories, tags)
* 404.php — 404 error page

Essential Theme Setup Hooks in functions.php:

* after\_setup\_theme  
  Hook to add theme support and register menus:

function mytheme\_setup() {

add\_theme\_support('title-tag');

add\_theme\_support('post-thumbnails');

add\_theme\_support('custom-logo');

add\_theme\_support('html5', ['search-form', 'comment-form', 'gallery']);

register\_nav\_menus([

'primary' => \_\_('Primary Menu', 'my-custom-theme'),

]);

}

add\_action('after\_setup\_theme', 'mytheme\_setup');

wp\_enqueue\_scripts  
Enqueue styles and scripts properly:

function mytheme\_scripts() {

wp\_enqueue\_style('mytheme-style', get\_stylesheet\_uri());

wp\_enqueue\_script('mytheme-main-js', get\_template\_directory\_uri() . '/js/main.js', ['jquery'], null, true);

}

add\_action('wp\_enqueue\_scripts', 'mytheme\_scripts');

### 2. **How do you override WooCommerce templates without breaking future updates?**

* Copy the WooCommerce template files you want to override from  
  wp-content/plugins/woocommerce/templates/
* Paste them inside your theme folder under:  
  your-theme/woocommerce/
* Modify these copies as needed.

Important notes:

* Keep track of WooCommerce updates and template changes via WooCommerce > Status > Templates.
* Use WooCommerce hooks and filters where possible instead of copying entire templates.
* Avoid modifying plugin core files directly.
* For small layout changes, use hooks like woocommerce\_before\_shop\_loop, woocommerce\_after\_shop\_loop\_item, etc.

### 3. **What are the best ways to enqueue scripts and styles in themes and plugins?**

* Always use WordPress functions: wp\_enqueue\_style() and wp\_enqueue\_script().
* Use hooks to enqueue scripts at the right time:
  + For themes: use wp\_enqueue\_scripts
  + For admin pages: use admin\_enqueue\_scripts
  + For login page: use login\_enqueue\_scripts

function mytheme\_enqueue\_assets() {

wp\_enqueue\_style('mytheme-style', get\_stylesheet\_uri(), [], '1.0');

wp\_enqueue\_script('mytheme-script', get\_template\_directory\_uri() . '/js/script.js', ['jquery'], '1.0', true);

}

add\_action('wp\_enqueue\_scripts', 'mytheme\_enqueue\_assets');

Best practices:

* Declare dependencies (e.g., ['jquery']) so WordPress loads them first.
* Use versioning to control cache.
* Load JavaScript in footer (true in last argument) when possible.
* Use wp\_register\_script() if you want to register but not enqueue immediately.
* Conditional enqueue scripts only on relevant pages to optimize performance.

### 4. **How do you make themes translation-ready?**

* In style.css, set the Text Domain in the header:

/\*

Text Domain: my-custom-theme

\*/

Wrap all user-facing strings with translation functions, for example:

* \_\_() — Returns translated string

\_\_('Hello World', 'my-custom-theme');

\_e() — Echo translated string

\_e('Hello World', 'my-custom-theme');

Load the text domain in your theme’s functions.php:

function mytheme\_load\_textdomain() {

load\_theme\_textdomain('my-custom-theme', get\_template\_directory() . '/languages');

}

add\_action('after\_setup\_theme', 'mytheme\_load\_textdomain');

Create .pot file using tools like Poedit or WP-CLI:

wp i18n make-pot . languages/my-custom-theme.pot

Translate .pot to .po and .mo files for each language and place them inside /languages folder.

### 1. **How to register custom post types and taxonomies with full REST API support?**

Use register\_post\_type() and register\_taxonomy() with 'show\_in\_rest' => true to enable REST API support.

Example: Register Custom Post Type

function myplugin\_register\_custom\_post\_type() {

$args = [

'label' => \_\_('Books', 'my-plugin'),

'public' => true,

'has\_archive' => true,

'show\_in\_rest' => true, // Enable REST API

'supports' => ['title', 'editor', 'thumbnail', 'custom-fields'],

'rewrite' => ['slug' => 'books'],

'menu\_icon' => 'dashicons-book',

];

register\_post\_type('book', $args);

}

add\_action('init', 'myplugin\_register\_custom\_post\_type');

Example: Register Custom Taxonomy

function myplugin\_register\_custom\_taxonomy() {

$args = [

'labels' => [

'name' => \_\_('Genres', 'my-plugin'),

'singular\_name' => \_\_('Genre', 'my-plugin'),

],

'public' => true,

'hierarchical' => true, // Like categories (true) or tags (false)

'show\_in\_rest' => true, // Enable REST API

'rewrite' => ['slug' => 'genre'],

];

register\_taxonomy('genre', ['book'], $args);

}

add\_action('init', 'myplugin\_register\_custom\_taxonomy');

### 2. **How do you add custom fields and meta boxes to custom post types?**

* Meta Boxes: Use the add\_meta\_box() function to add custom meta boxes in the post editor.

Example adding meta box:

function myplugin\_add\_meta\_boxes() {

add\_meta\_box(

'myplugin\_book\_details',

\_\_('Book Details', 'my-plugin'),

'myplugin\_book\_details\_callback',

'book', // post type slug

'normal',

'high'

);

}

add\_action('add\_meta\_boxes', 'myplugin\_add\_meta\_boxes');

function myplugin\_book\_details\_callback($post) {

wp\_nonce\_field('myplugin\_save\_book\_details', 'myplugin\_book\_nonce');

$author = get\_post\_meta($post->ID, '\_book\_author', true);

echo '<label for="book\_author">' . \_\_('Author', 'my-plugin') . '</label>';

echo '<input type="text" id="book\_author" name="book\_author" value="' . esc\_attr($author) . '" size="25" />';

}

Saving Meta Box Data:

function myplugin\_save\_book\_details($post\_id) {

if (!isset($\_POST['myplugin\_book\_nonce']) || !wp\_verify\_nonce($\_POST['myplugin\_book\_nonce'], 'myplugin\_save\_book\_details')) {

return;

}

if (defined('DOING\_AUTOSAVE') && DOING\_AUTOSAVE) {

return;

}

if (!current\_user\_can('edit\_post', $post\_id)) {

return;

}

if (isset($\_POST['book\_author'])) {

update\_post\_meta($post\_id, '\_book\_author', sanitize\_text\_field($\_POST['book\_author']));

}

}

add\_action('save\_post', 'myplugin\_save\_book\_details');

Custom Fields with Gutenberg:  
Use the register\_post\_meta() function with 'show\_in\_rest' => true to expose custom fields in the REST API and Gutenberg editor.

function myplugin\_register\_post\_meta() {

register\_post\_meta('book', '\_book\_author', [

'show\_in\_rest' => true,

'single' => true,

'type' => 'string',

'auth\_callback' => function() {

return current\_user\_can('edit\_posts');

},

]);

}

add\_action('init', 'myplugin\_register\_post\_meta');

### **3. What is the difference between post meta and custom tables? When would you use custom tables?**

| Aspect | Post Meta | Custom Tables |
| --- | --- | --- |
| Storage | Stored in wp\_postmeta table, key-value pairs linked to post ID | Custom database tables with custom schema |
| Performance | Good for small/medium metadata, but can be slow for large datasets or complex queries | Better for large data sets, complex relationships, or performance critical data |
| Querying | Limited querying options, no complex joins; uses meta\_query but can be inefficient | Full control over schema and queries; can optimize indexes and relations |
| Complexity | Simple to implement; WordPress built-in | Requires custom SQL, migrations, and maintenance |
| Use Cases | Storing simple meta like custom fields, small sets of data | Handling complex data structures, reports, logs, or data unrelated to posts |

When to use custom tables?

* When your data structure is complex and not suited for simple key-value pairs (e.g., large tabular data, logs, analytics).
* When you need optimized queries or relational data with multiple joins.
* When data size or performance becomes a bottleneck with post meta.

When to stick with post meta?

* When you store simple metadata related directly to posts.
* When you want easy integration with WP functions like get\_post\_meta(), REST API, and Gutenberg.

### **1. How do you customize the WooCommerce checkout process with custom fields?**

You can add custom fields to the checkout page by hooking into WooCommerce actions and filters.

Example: Add a custom text field (e.g., “Delivery Instructions”)

* Add field to checkout form

add\_action('woocommerce\_after\_order\_notes', 'my\_custom\_checkout\_field');

function my\_custom\_checkout\_field($checkout) {

echo '<div id="my\_custom\_checkout\_field"><h2>' . \_\_('Delivery Instructions') . '</h2>';

woocommerce\_form\_field('delivery\_instructions', [

'type' => 'textarea',

'class' => ['form-row-wide'],

'label' => \_\_('Please provide any delivery instructions'),

'placeholder' => \_\_('Leave instructions here'),

], $checkout->get\_value('delivery\_instructions'));

echo '</div>';

}

Validate the field (optional)

add\_action('woocommerce\_checkout\_process', 'my\_custom\_checkout\_field\_process');

function my\_custom\_checkout\_field\_process() {

if (isset($\_POST['delivery\_instructions']) && empty(trim($\_POST['delivery\_instructions']))) {

wc\_add\_notice(\_\_('Please provide delivery instructions.'), 'error');

}

}

Save the field data to order meta

add\_action('woocommerce\_checkout\_update\_order\_meta', 'my\_custom\_checkout\_field\_update\_order\_meta');

function my\_custom\_checkout\_field\_update\_order\_meta($order\_id) {

if (!empty($\_POST['delivery\_instructions'])) {

update\_post\_meta($order\_id, '\_delivery\_instructions', sanitize\_textarea\_field($\_POST['delivery\_instructions']));

}

}

Display field value in admin order edit page

add\_action('woocommerce\_admin\_order\_data\_after\_billing\_address', 'my\_custom\_checkout\_field\_display\_admin\_order\_meta', 10, 1);

function my\_custom\_checkout\_field\_display\_admin\_order\_meta($order) {

$instructions = get\_post\_meta($order->get\_id(), '\_delivery\_instructions', true);

if ($instructions) {

echo '<p><strong>' . \_\_('Delivery Instructions') . ':</strong> ' . esc\_html($instructions) . '</p>';

}

}

### **2. Explain how to override WooCommerce templates and extend hooks**

* Overriding Templates:
  + Copy the template file from WooCommerce plugin directory:  
    wp-content/plugins/woocommerce/templates/checkout/form-checkout.php
  + Paste it in your theme at:  
    wp-content/themes/your-theme/woocommerce/checkout/form-checkout.php
  + Modify the copied template as needed.
* Extend WooCommerce using hooks (actions and filters):
  + WooCommerce has many actions and filters to customize behavior without overriding entire templates.
  + Example of adding content before the cart totals:

add\_action('woocommerce\_cart\_totals\_before\_order\_total', function() {

echo '<p>Free shipping on orders over $100!</p>';

});

Use filters to modify output, e.g., to change the “Add to cart” button text:

add\_filter('woocommerce\_product\_single\_add\_to\_cart\_text', function() {

return \_\_('Buy Now', 'my-theme');

});

### **3. How do you create dynamic pricing or discounts programmatically?**

Use WooCommerce cart hooks to adjust prices or apply discounts dynamically.

Example: Apply 10% discount for logged-in users

add\_action('woocommerce\_cart\_calculate\_fees', 'my\_custom\_dynamic\_discount');

function my\_custom\_dynamic\_discount($cart) {

if (is\_admin() && !defined('DOING\_AJAX')) return;

if (is\_user\_logged\_in()) {

$discount = $cart->subtotal \* 0.10;

$cart->add\_fee(\_\_('Loyalty Discount', 'my-theme'), -$discount);

}

}

Example: Change product price dynamically

add\_action('woocommerce\_before\_calculate\_totals', 'my\_custom\_price\_adjustment', 20);

function my\_custom\_price\_adjustment($cart) {

if (is\_admin() && !defined('DOING\_AJAX')) return;

foreach ($cart->get\_cart() as $cart\_item) {

$product = $cart\_item['data'];

// Give 20% off on product with ID 123

if ($product->get\_id() == 123) {

$new\_price = $product->get\_price() \* 0.8;

$product->set\_price($new\_price);

}

}

}

### **4. How would you create custom WooCommerce REST API endpoints?**

* Register a custom REST API route using WordPress REST API infrastructure.

Example: Add a custom WooCommerce REST endpoint to get all orders over $100

add\_action('rest\_api\_init', function() {

register\_rest\_route('myplugin/v1', '/high-value-orders', [

'methods' => 'GET',

'callback' => 'myplugin\_get\_high\_value\_orders',

'permission\_callback' => function() {

return current\_user\_can('manage\_woocommerce');

}

]);

});

function myplugin\_get\_high\_value\_orders(WP\_REST\_Request $request) {

$args = [

'limit' => -1,

'status' => ['wc-completed', 'wc-processing'],

'min\_total' => 100,

];

// Using WC REST API classes to query orders

$orders = wc\_get\_orders($args);

$data = [];

foreach ($orders as $order) {

$data[] = [

'id' => $order->get\_id(),

'total' => $order->get\_total(),

'date' => $order->get\_date\_created()->date('Y-m-d H:i:s'),

'customer' => $order->get\_billing\_first\_name() . ' ' . $order->get\_billing\_last\_name(),

];

}

return rest\_ensure\_response($data);

}

This endpoint will be available at:  
/wp-json/myplugin/v1/high-value-orders

* Use proper permissions and security checks in the permission\_callback.

### **1. What caching strategies do you use in WordPress (object cache, page cache, CDN)?**

* Object Cache:  
  Caches database query results in memory to reduce repeated queries during a single page load or across requests (if persistent).
  + Use built-in WordPress Object Cache APIs with persistent backends like Redis or Memcached.
  + Example: Redis cache with the redis-cache plugin or custom integration.
* Page Cache:  
  Stores fully rendered HTML pages and serves them directly without running PHP or hitting the database.
  + Plugins like WP Rocket, W3 Total Cache, or server-level cache like Varnish or Nginx FastCGI cache.
  + Good for high-traffic, mostly static sites.
* CDN (Content Delivery Network):  
  Distributes static assets (images, CSS, JS) geographically to reduce load times globally.
  + Popular CDNs: Cloudflare, AWS CloudFront, BunnyCDN.
  + Can also cache entire pages if integrated properly.
* Additional:
  + Opcode caching (e.g., OPcache) at the PHP engine level.
  + Browser caching via HTTP headers.
  + Query caching plugins for frequently run queries.

### 2. **How do you secure a WordPress site and plugins from common vulnerabilities?**

* Keep WordPress, themes, and plugins updated regularly to patch known vulnerabilities.
* Use strong passwords and two-factor authentication (2FA) for all admin users.
* Restrict admin access by IP or use plugins like Wordfence or Sucuri for firewall and malware scanning.
* Disable file editing in WordPress admin by adding define('DISALLOW\_FILE\_EDIT', true); in wp-config.php.
* Use least privilege principle — assign proper roles/capabilities, avoid admin access if not necessary.
* Sanitize, validate, and escape all user inputs and outputs (see below).
* Use HTTPS everywhere with valid SSL certificates.
* Implement security headers (CSP, HSTS, X-Frame-Options, etc.).
* Limit login attempts and protect XML-RPC (disable if unused).
* Backup site regularly and store backups securely.

### 3. **How do you prevent SQL injection, XSS, and CSRF in your code?**

* Prevent SQL Injection:
  + Use WordPress $wpdb->prepare() for all database queries with dynamic data.
  + Avoid direct SQL concatenation.
  + Example:

global $wpdb;

$user\_id = intval($\_GET['user\_id']);

$result = $wpdb->get\_results($wpdb->prepare("SELECT \* FROM $wpdb->users WHERE ID = %d", $user\_id));

Prevent Cross-Site Scripting (XSS):

* Always escape output using functions like esc\_html(), esc\_attr(), esc\_url() depending on context.
* Use wp\_kses() to allow limited HTML where necessary.
* Sanitize input with sanitize\_text\_field(), sanitize\_email(), sanitize\_textarea\_field(), etc.
* Prevent Cross-Site Request Forgery (CSRF):
* Use WordPress nonces (wp\_nonce\_field(), check\_admin\_referer()) in forms and AJAX requests to verify request authenticity.
* Example:

if (!isset($\_POST['my\_nonce']) || !wp\_verify\_nonce($\_POST['my\_nonce'], 'my\_action')) {

wp\_die('Security check failed');

}

### **4. How do you optimize database queries in WordPress?**

* Use WP\_Query efficiently by limiting returned fields, avoiding posts\_per\_page => -1 unless necessary.
* Cache expensive query results using transients API or object cache.
* Add proper indexes to custom tables or meta fields if you create them.
* Avoid N+1 query problem — batch queries instead of querying inside loops.
* Use get\_posts() or WP\_Query with 'fields' => 'ids' if only IDs are needed.
* Offload heavy data processing to background jobs or async requests if possible.
* Avoid or optimize queries on wp\_postmeta as it can get large and slow; consider custom tables for heavy meta usage.
* Monitor and analyze queries with tools like Query Monitor plugin or by enabling MySQL slow query log.
* Delete orphaned post meta, transients, and revisions regularly to keep tables clean.























































